Javaの イケてるコード 残念なコード

読みやすいJavaコードを書く技術

Acroquest Technology
阪本 雄一郎

本連載では、Javaプログラミングの改善法やテクニックについて解説していきます。「Javaの入門書は読んだけど次に読むべき本がわからない」、「研修や仕事でプログラムは書くようになったけど、もっとレベルアップしたい」という人はぜひ参考にしてください。

良いコード、悪いコードとは?

Javaの良いコード、悪いコードとは何でしょうか?一言で「良い」と言っても「読みやすい」「修正しやすい」「無駄な処理時間が掛からない」「メモリーを無駄遣いしない」など様々な評価基準があります。

まずは、**リスト1**のコードを見てください。このコードは"読みやすい"でしょうか。引数が多くifのネストが深い、変数の意味がわかりにくい、などおそらく多くの人は、読みにくい、と感じるでしょう。このコードを読み解くには、引数の意味を理解し、多くのif文の条件をすべて読み解かなくてはなりません。「読みにくいコード」は、典型的な「悪いコード」だと筆者は考えています。逆に、「読みやすいコード」は、良いコードの一例でしょう。

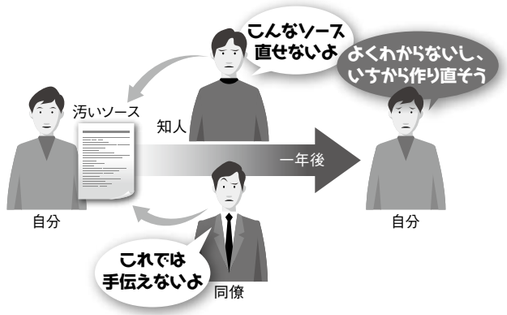

図1●汚いソースでは他人に迷惑がかかるだけではなく自分も読めなくなる

リスト1●典型的な悪いコード例

```
public void cnt(String line, int lines, int comments,
  int empty, boolean comment, boolean flag) {
  for (int i = 0; i < line.length(); i++) {
    if (comment) {
      if (i + 1 < line.length()) {
        if ("*/".equals(line.substring(i, i+2))) {
          comment = false;
        }
      }
    } else if (flag) {
      if (line.charAt(i) == '¥¥') {
        if (i + 1 < line.length()) {
          i++;
        }
      } else if (line.charAt(i) == '"')
      flag = false;
      } else {
        if (i + 1 < line.length() && "/*".equals(
          line.substring(i, i+2))) {
            // commentをtrueにする
            comment = true;
        }
        if (i + 1 < line.length() && "//".equals(
          line.substring(i, i+2)))
        break;
      if (line.charAt(i) == '"') {
        if (flag) flag = false;
        else flag = true;
      }
      ()
```

名前の意味がわからず処理内容が不明
引数が多すぎてメソッド利用時に困る
ifのネストが深い
インデントがそろっていないのでifの範囲がわからない
当たり前のコメントはいらない
一般的過ぎる名前のため用途がわからない

読みにくいコードは、何が悪いのでしょうか。読みにくいコードの一番の問題は、後で手を加えることが難しい、ということです。自分が読みにくいコードを書けば、同僚や知人は自分の作るアプリケーションに機能を追加したり、不具合を修正したりすることを嫌がります。時間も掛かるでしょう(**図1**)。

修正を嫌がるのは、同僚や知人だけではありません。あなた自身も嫌になるはずです。例えば、あなたの書いた3日前のコードをすべて覚えていますか。3日程度なら覚えているかもしれませんが、1週間、1カ月、さらには1年経った後では、ほとんどの人は忘れてしまいます。「3カ月も経てば他人のコード」と言われるように、読みにくいコードは自分自身にも読みにくいコードになるのです。

リスト2●ラインカウンタのプログラムの一部。あなたはこのコードを読めますか

```
public class LineCounter {
  public static final String FILE_NAME
    = "Sample.java";

  public static void main(String[] args) {
    new LineCounter().count();
  }

  public void count() {
    String fileName = "Sample.java";
    boolean nest = false;
    BufferedReader reader = null;
    int lines = 0;
    int steps = 0;
    int comments = 0;
    int empty = 0;
    boolean inSingleQuote = false;
    boolean inDubleQuote = false;
    String line = null;
    boolean comment = false;
    boolean flag = false;
    boolean flag2 = false;
    int commentNest = 0;
    try {
      reader = new BufferedReader(
        new FileReader(fileName));
      while ((line = reader.readLine()) != null) {
        flag = false;
        flag2 = false;
        for (int i = 0; i < line.length(); i++) {
          if (line.charAt(i) != ' '
            && line.charAt(i) != '¥t') {
            if (comment) {
              flag2 = true;
              if (i + 1 < line.length()
                && "*/".equals(line.substring(i, i + 2))) {
                if (nest) {
                  commentNest--;
                  if (commentNest == 0) {
                    comment = false;
                  }
                } else {
                  comment = false;
                }
                i++;
              } else if (i + 1 < line.length()
                && "/*".equals(line.substring(i, i + 2))) {
                if (nest) {
                  commentNest++;
                }
                i++;
              }
            } else if (inSingleQuote) {
              flag = true;
              if (line.charAt(i) == '¥¥') {
                i++;
              } else if (line.charAt(i) == '¥'') {
                inSingleQuote = false;
              }
            } else if (inDubleQuote) {
              flag = true;
              if (line.charAt(i) == '¥¥') {
                i++;
              } else if (line.charAt(i) == '"') {
                inDubleQuote = false;
              }
            } else {
              if (i + 1 < line.length() &&
                "/*".equals(line.substring(i, i + 2))) {
                // commentをtrueにする
                comment = true;
                flag2 = true;
                i++;
                if (nest) {
                  commentNest++;
                }
              } else if (i + 1 < line.length() &&
                "//".equals(line.substring(i, i + 2))) {
                flag2 = true;
                break;
              } else {
                flag = true;
                if (line.charAt(i) == '"') {
                  inDubleQuote = true;
                } else if (line.charAt(i) == '¥'') {
                  inSingleQuote = true;
                }
              }
            }
          }
          lines++;
          if (flag) {
            steps++;
          } else if (flag2 || comment) {
            comments++;
          } else {
            empty++;
          }
        }
        System.out.println("行数:" + lines);
        System.out.println("ステップ数:" + steps);
        System.out.println("コメント数:" + comments);
        System.out.println("空行数:" + empty);
      } catch (IOException ex) {
        System.err.println(
          fileName + "の読み込み中にエラーが発生しました。");
        ex.printStackTrace();
      } finally {
        if (reader != null) {
          try {
            reader.close();
          } catch (IOException ex) {
            ex.printStackTrace();
          }
        }
      }
    }
  }
```

ポイント1 変数の名前が一般的過ぎて理解できない
ポイント2 ネストが深い
ポイント3 条件を反転させてネストを減らせる
ポイント4 変数はただ列挙するのではなく、使う直前に宣言する
ポイント5 頻繁に呼び出すものは一時変数にまとめる
ポイント6 無意味なコメントは書かない

これぞ残念なコード

それでは、どうすれば「良いコード」を書けるのでしょう。今回は「読みやすいコードを書く」ことを目的に、コードを改善する方法を説明します。

リスト2は、Javaのソースファイルのステップ数やコメン